Chapter 18 - Additional Material

Section 18.1 - Do-while loops

A ***do-while loop*** is a loop construct that first executes the loop body's statements, then checks the loop condition.

Construct 18.1.1: Do-while loop.

```
do {
   // Loop body
} while (loopExpression);
```

Versus a while loop, a do-while loop is useful when the loop should iterate at least once.

P Participation Activity

18.1.1: Do-while loop.

Start

```
import java.util.Scanner;

public class DoWhile {
   public static void main(String[] args) {
      String fill = "*";
      Scanner in = new Scanner(System.in);

      do {
         System.out.print("\n");
         System.out.println(fill + fill + fill);
         System.out.println(fill + fill + fill);
         System.out.println(fill + fill + fill);
         System.out.print("Enter char (q to quit): ");
         fill = in.next();
      } while (!fill.equals("q"));

      return;
   }
}
```

```
***
***
***
Enter char (q to quit: )  X

XXX
XXX
XXX
Enter char (q to quit: )  q
```

(program done)

P Participation Activity

18.1.2: Do-while loop.

Consider the following loop:

```
int count = 0;
int num = 6;
do {
   num = num - 1;
   count = count + 1;
} while (num > 4);
```

#	Question	Your answer
1	What is the value of count after the loop?	0
		1
		2
2	What initial value of num would prevent count from being incremented?	4
		0
		No such value.

C Challenge Activity

18.1.1: Basic do-while loop with user input.

Complete the do-while loop to output 0 to countLimit. Assume the user will only input a positive num

```
public class CountToLimit {
   public static void main (String [] args) {
      Scanner scnr = new Scanner(System.in);
      int countLimit = 0;
      int printVal = 0;

      // Get user input
      countLimit = scnr.nextInt();

      printVal = 0;
      do {
         System.out.print(printVal + " ");
         printVal = printVal + 1;
      } while ( /* Your solution goes here  */ );
      System.out.println("");

      return;
   }
}
```

Run

C Challenge Activity

18.1.2: Do-while loop to prompt user input.

Write a do-while loop that continues to prompt a user to enter a number less than 100, until the ente
End each prompt with newline Ex: For the user input 123, 395, 25, the expected output is:

```
Enter a number (<100):
Enter a number (<100):
Enter a number (<100):
Your number < 100 is: 25
```

```java
import java.util.Scanner;

public class NumberPrompt {
   public static void main (String [] args) {
   Scanner scnr = new Scanner(System.in);
   int userInput = 0;

      /* Your solution goes here  */

      System.out.println("Your number < 100 is: " + userInput);

      return;
   }
}
```

Run

Section 18.2 - Engineering examples

Arrays can be useful in solving various engineering problems. One problem is computing the voltage drop across a series of resistors. If the total voltage across the resistors is V, then the current through the resistors will be I = V/R, where R is the sum of the resistances. The voltage drop Vx across resistor x is then Vx = I · Rx. The following program uses an array to store a user-entered set of resistance values, computes I, then computes the voltage drop across each resistor and stores each in another array, and finally prints the results.

Figure 18.2.1: Calculate voltage drops across series of resistors.

```
import java.util.Scanner;

public class ResistorVoltage {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_RES = 5;                    // Number of resistors
      double[] resVals = new double[NUM_RES];   // Ohms
      double circVolt = 0;                      // Volts
      double[] vDrop = new double[NUM_RES];     // Volts
      double currentVal = 0;                    // Amps
      double sumRes = 0;                        // Ohms
      int i = 0;                                // Loop index

      System.out.println("5 resistors are in series.");
      System.out.println("This program calculates the");
      System.out.println("voltage drop across each resistor.\n")

      System.out.print("Input voltage applied to circuit: ");
      circVolt = scnr.nextDouble();

      System.out.println("Input ohms of " + NUM_RES + " resistor
      for (i = 0; i < NUM_RES; ++i) {
         System.out.print((i + 1) + ") ");
         resVals[i] = scnr.nextDouble();
      }

      // Calculate current
      for (i = 0; i < NUM_RES; ++i) {
         sumRes = sumRes + resVals[i];
      }
      currentVal = circVolt / sumRes;     // I = V/R

      for (i = 0; i < NUM_RES; ++i) {
         vDrop[i] = currentVal * resVals[i]; // V = IR
      }

      System.out.println("\nVoltage drop per resistor is:");
      for (i = 0; i < NUM_RES; ++i) {
         System.out.println((i + 1) + ") "
                 + "" + vDrop[i] + " V");
      }

      return;
   }
}
```

```
5 resistors are in
This program calcu
voltage drop acros

Input voltage appl
Input ohms of 5 re
1) 3.3
2) 1.5
3) 2
4) 4
5) 2.2

Voltage drop per re
1) 3.0461538461538
2) 1.3846153846153
3) 1.8461538461538
4) 3.6923076923076
5) 2.0307692307692
```

P Participation Activity

18.2.1: Voltage drop program.

#	Question	Your answer
1	What does variable circVolt store?	Multiple voltages, one for each resistor.
		The resistance of each resistor.
		The total voltage across the series of resistors.
2	What does the first for loop do?	Gets the voltage of each resistor and stores each in an array.
		Gets the resistance of each resistor and stores each in an array.
		Adds the resistances into a total value.
3	What does the second for loop do?	Adds the resistances into a single value, so that I = V/R can be computed.
		Computes the voltage across each resistor.
4	What does the third for loop do?	Update the resistances array with new resistor values.
		Sum the voltages across each resistor into a total voltage.
		Determines the voltage drop across each resistor and stores each voltage in another array.
	Could the fourth loop's statement have been incorporated into the third loop, thus eliminating the fourth loop?	No, a resistor's voltage drop isn't

5	into the third loop, thus eliminating the fourth loop?	voltage drop isn't known until the entire loop has finished.
		Yes, but keeping the loops separate is better style.

Engineering problems commonly involve matrix representation and manipulation. A matrix can be captured using a two-dimensional array. Then matrix operations can be defined on such arrays. The following illustrates matrix multiplication for 4x2 and 2x3 matrices captured as two-dimensional arrays.

Figure 18.2.2: Matrix multiplication of 4x2 and 2x3 matrices.

```
public class MatrixMult {

   public static void main(String[] args) {
      final int M1_ROWS = 4;        // Matrix 1 rows
      final int M1_COLS = 2;        // Matrix 2 cols
      final int M2_ROWS = M1_COLS; // Matrix 2 rows (must have same value)
      final int M2_COLS = 3;        // Matrix 2 cols
      int rowIndex = 0;             // Loop index
      int colIndex = 0;             // Loop index
      int elemIndex = 0;            // Loop index
      int dotProd = 0;              // Dot product

      // M1_ROWS by M1_COLS
      int[][] m1 = {{3, 4},
                    {2, 3},
                    {1, 5},
                    {0, 2}};

      // M2_ROWS by M2_COLS
      int[][] m2 = {{5, 4, 4},
                    {0, 2, 3}};

      // M1_ROWS by M2_COLS
      int[][] m3 = {{0, 0, 0},
                    {0, 0, 0},
                    {0, 0, 0},
                    {0, 0, 0}};

      // m1 * m2 = m3
      for (rowIndex = 0; rowIndex < M1_ROWS; ++rowIndex) {
         for (colIndex = 0; colIndex < M2_COLS; ++colIndex) {
            // Compute dot product
            dotProd = 0;
            for (elemIndex = 0; elemIndex < M2_ROWS; ++elemIndex) {
               dotProd = dotProd + (m1[rowIndex][elemIndex] * m2[elemIndex][colInde
            }

            m3[rowIndex][colIndex] = dotProd;
         }
      }

      // Print m3 result
      for (rowIndex = 0; rowIndex < M1_ROWS; ++rowIndex) {
         for (colIndex = 0; colIndex < M2_COLS; ++colIndex) {
            System.out.print(m3[rowIndex][colIndex] + " ");
         }
         System.out.println();
      }

      return;
   }
}
```

```
15 20 24
10 14 17
5 14 19
0 4 6
```

P Participation Activity

18.2.2: Matrix multiplication program.

#	Question	Your answer
1	For the first set of for loops, how many dot products are computed? (In other words, how many iterations are due to the outer two for loops?)	
2	For the first set of for loops, the inner-most loop computes a dot product. Each time that inner-most loop is reached, how many times will it iterate?	

Section 18.3 - Engineering examples using methods

This section contains examples of methods for various engineering calculations.

Gas equation

An equation used in physics and chemistry that relates pressure, volume, and temperature of a gas is PV = nRT. P is the pressure, V the volume, T the temperature, n the number of moles, and R a constant. The method below outputs the temperature of a gas given the other values.

Figure 18.3.1: PV = nRT. Compute the temperature of a gas.

```java
import java.util.Scanner;

public class GasTemperature {

   final static double GAS_CONSTANT = 8.3144621; // J / (mol*K)

   /* Converts a pressure, volume, and number of moles
    of a gas to a temperature. */
   public static double pvnToTemp(double gasPressure, double gasVolume,
                                  double numMoles) {
      return (gasPressure * gasVolume) / (numMoles * GAS_CONSTANT);
   }

   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      double gasPress = 0.0;  // User defined pressure
      double gasVol = 0.0;    // User defined volume
      double gasMoles = 0.0;  // User defined moles

      // Prompt user for input parameteres
      System.out.print("Enter pressure (in Pascals): ");
      gasPress = scnr.nextDouble();

      System.out.print("Enter volume (in cubic meters): ");
      gasVol = scnr.nextDouble();

      System.out.print("Enter number of moles: ");
      gasMoles = scnr.nextDouble();

      // Call method to calculate temperature
      System.out.print("Temperature = ");
      System.out.println(pvnToTemp(gasPress, gasVol, gasMoles) + " K");

      return;
   }
}
```

```
Enter pressure
Enter volume
Enter number
Temperature =
```

P Participation Activity

18.3.1: PV = nRT calculation.

Questions refer to pVnToTemp() above.

#	Question	Your answer
1	pVnToTemp() uses a rewritten form of PV = nRT to solve for T, namely T = PV/nR.	True False
2	pVnToTemp() uses a constant variable for the gas constant R.	True False
3	tempVolMolesToPressure() would likely return (temp * vlm) / (mols * GAS_CONSTANT).	True False

Projectile location

Common physics equations determine the x and y coordinates of a projectile object at any time, given the object's initial velocity and angle at time 0 with initial position x = 0 and y = 0. The equation for x is v * t * cos(a). The equation for y is v * t * sin(a) - 0.5 * g * t * t. The following provides a single method to compute an object's position; because position consists of two values (x and y), the method uses two array parameters to return values for x and y. The program's main method asks the user for the object's initial velocity, angle, and height (y position), and then prints the object's position for every second until the object's y position is no longer greater than 0 (meaning the object fell back to earth).

Figure 18.3.2: Trajectory of object on Earth.

```
import java.util.Scanner;

// Note: 1-letter variable names are typically avoided,
// but used below where standard in physics.
public class ObjectTrajectory {
   final static double PI_CONST = 3.14159265;
```

```
   final static double PI_CONST = 3.14159265;

   // Given time, angle, velocity, and gravity
   // Update x and y values
   public static void objectTrajectory(double t, double a, double v,
           double g, double[] x, double[] y) {
      x[0] = v * t * Math.cos(a);
      y[0] = v * t * Math.sin(a) - 0.5 * g * t * t;
      return;
   }

   // convert degree value to radians
   public static double degToRad(double deg) {
      return ((deg * PI_CONST) / 180.0);
   }

   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final double GRAVITY = 9.8;      // Earth gravity (m/s^2)
      double launchAngle = 0.0;        // Angle of launch (rad)
      double launchVelocity = 0.0;     // Velocity (m/s)
      double elapsedTime = 1.0;        // Time (s)

      double[] xLoc = new double[1];  // Object's height above ground (m)
      double[] yLoc = new double[1];  // Object's'horiz. dist. from start (m)

      xLoc[0] = -1.0;
      yLoc[0] = 0.0;

      System.out.print("Launch angle (deg): ");
      launchAngle = scnr.nextDouble();
      launchAngle = degToRad(launchAngle);  // To radians

      System.out.print("Launch velocity (m/s): ");
      launchVelocity = scnr.nextDouble();

      System.out.print("Initial height (m): ");
      yLoc[0] = scnr.nextDouble();

      while (yLoc[0] > 0.0) {  // While above ground
         System.out.println("Time " + elapsedTime + "   x = " + xLoc[0]
                 + "   y = " + yLoc[0]);
         objectTrajectory(elapsedTime, launchAngle, launchVelocity,
                          GRAVITY, xLoc, yLoc);
         elapsedTime = elapsedTime + 1.0;
      }

      return;
   }
}
```

```
Launch angle (deg): 45
Launch velocity (m/s): 100
Initial height (m): 3
Time 1.0   x = -1.0   y = 3.0
Time 2.0   x = 70.71067818211394   y = 65.81067805519557
Time 3.0   x = 141.42135636422788   y = 121.82135611039115
Time 4.0   x = 212.13203454634183   y = 168.03203416558674
Time 5.0   x = 282.84271272845575   y = 204.44271222078228
Time 6.0   x = 353.5533909105697   y = 231.05339027597785
Time 7.0   x = 424.26406909268366   y = 247.86406833117346
Time 8.0   x = 494.97474727479755   y = 254.874746386369
```

```
Time 9.0   x = 565.6854254569115   y = 252.08542444156456
Time 10.0   x = 636.3961036390255   y = 239.4961024967601
Time 11.0   x = 707.1067818211394   y = 217.1067805519557
Time 12.0   x = 777.8174600032534   y = 184.91745860715127
Time 13.0   x = 848.5281381853673   y = 142.9281366623469
Time 14.0   x = 919.2388163674813   y = 91.13881471754246
Time 15.0   x = 989.9494945495951   y = 29.54949277273795
```

P Participation Activity

18.3.2: Projective location.

Questions refer to objectTrajectory() above.

#	Question	Your answer
1	objectTrajectory() cannot return two values (for x and y), so instead takes x and y as modifiable parameters and changes their values.	True False
2	objectTrajectory() could replace double types by int types without causing much change in computed values.	True False
3	Each iteration of the loop will see yLoc increase.	True False
4	Assuming the launch angle is less than 90 degrees, each iteration of the loop will see xLoc increase.	True False

C Challenge Activity

18.3.1: Method to compute gas volume.

Define a method computeGasVolume that returns the volume of a gas given parameters pressure, te
equation PV = nRT, where P is pressure in Pascals, V is volume in cubic meters, n is number of mole
/ (mol*K)), and T is temperature in Kelvin. All parameter types and the return type are double.

```java
   /* Your solution goes here  */

   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      double gasPressure = 0.0;
      double gasMoles = 0.0;
      double gasTemperature = 0.0;
      double gasVolume = 0.0;

      gasPressure = 100;
      gasMoles = 1 ;
      gasTemperature = 273;

      gasVolume = computeGasVolume(gasPressure, gasTemperature, gasMoles);
      System.out.println("Gas volume: " + gasVolume + " m^3");

      return;
   }
}
```

Run

Section 18.4 - Command-line arguments

Command-line arguments are values entered by a user when running a program from a command line. A *command line* exists in some program execution environments, wherein a user types a program's name and any arguments at a command prompt. To access those arguments, main() can be defined with a special parameter args, as shown below. The program prints provided command-line arguments. (The "for" loop is not critical to understanding the point, in case you haven't studied for loops yet).

Figure 18.4.1: Printing command-line arguments.

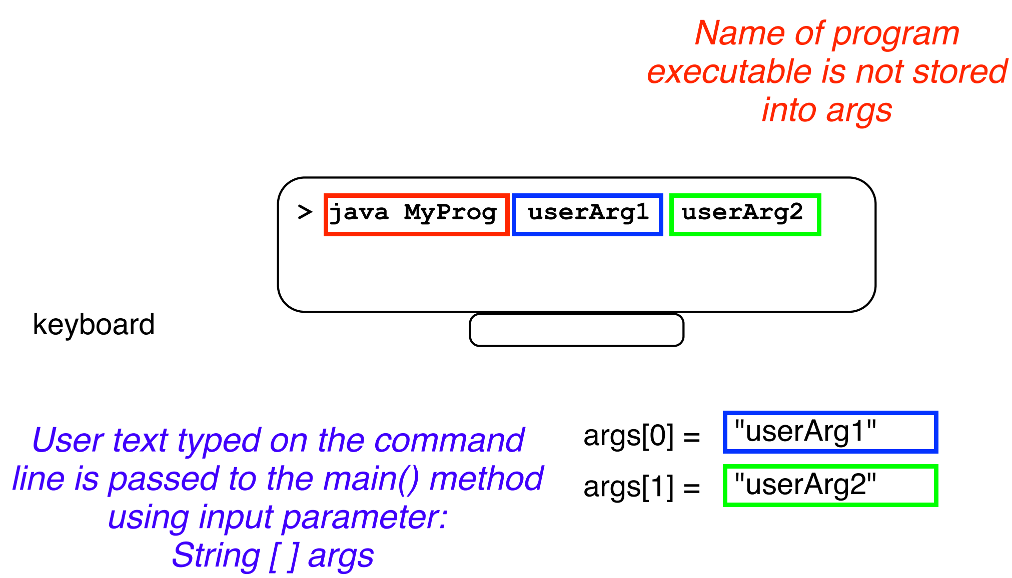
```
public class ArgTest {
   public static void main(String[] args) {
      int i = 0;
      int argc = args.length;

      System.out.println("args.length: " + argc);

      for (i = 0; i < argc; ++i) {
         System.out.println("args[" + i + "]: " + a
      }

      return;
   }
}
```

```
> java ArgTest
args.length: 0

> java ArgTest Hello
args.length: 1
args[0]: Hello

> java ArgTest Hey ABC 99 -5
args.length: 4
args[0]: Hey
args[1]: ABC
args[2]: 99
args[3]: -5
```

Then, when a program is run, the system passes the parameter ***args*** to main(), defined as an array of Strings. args is known as the arguments array and has one String for each command-line argument. A program can determine the number of command-line arguments by accessing args' length field, as in: int argc = args.length;.

P Participation Activity

18.4.1: Command-line arguments.



P Participation Activity

18.4.2: Command-line arguments.

#	Question	Your answer
1	What is args.length for: `java MyProg 13 14 smith`	
2	What is the value of args.length for: `java MyProg 12:55 PM`	
3	What is the string in args[1] for: `java MyProg Jan Feb Mar`	

The following program, named NameAgeParser, expects two command-line arguments.

Figure 18.4.2: Simple use of command-line arguments.

```
public class NameAgeParser {
   public static void main(String[] args) {
      String nameStr = "";  // User name
      String ageStr  = "";  // User age

      // Get inputs from command line
      nameStr = args[0];
      ageStr  = args[1];

      // Output result
      System.out.print("Hello " + nameStr + ". ")
      System.out.println(ageStr + " is a great ag

      return;
   }
}
```

```
> java NameAgeParser Amy 12
Hello Amy. 12 is a great age.

> java NameAgeParser Rajeev 44 HEY
Hello Rajeev. 44 is a great age.

> java NameAgeParser Denming
Exception in thread "main"
      java.lang.ArrayIndexOutOfBou
        at NameAgeParser.main(Name
```

However, there is no guarantee a user will type two command-line arguments. Extra arguments, like "HEY" above, are ignored. Conversely, too few arguments can cause a problem. In particular, a common error is to access elements in args without first checking args.length to ensure the user entered enough arguments, resulting in an out-of-range array access. In the last run above, the user typed too few arguments, causing an out-of-range array access.

When a program uses command-line arguments, good practice is to check args.length for the correct number of arguments. If the number of command-line arguments is incorrect, good practice is to print a usage message. A ***usage message*** lists a program's expected command-line arguments.

Figure 18.4.3: Checking for proper number of command-line arguments.

```
public class NameAgeParser {
   public static void main(String[] args) {
      String nameStr = "";  // User name
      String ageStr  = "";  // User age

      // Check if correct number of arguments provided
      if (args.length != 2) {
         System.out.println("Usage: java NameAgeParser name age");
         return;
      }

      // Grab inputs from command line
      nameStr = args[0];
      ageStr = args[1];

      // Output result
      System.out.print("Hello " + nameStr + ". ");
      System.out.println(ageStr + " is a great age.");

      return;
   }
}
```

```
> java NameAgePa
Hello Amy. 12 i

...

> java NameAgePa
Usage: myprog.e

...

> java NameAgePa
Usage: myprog.e
```

P Participation Activity

18.4.3: Checking the number of command-line arguments.

#	Question	Your answer
1	If a user types the wrong number of command-line arguments, good practice is to print a usage message.	True False
2	If a user types too many arguments but a program doesn't check for that, the program typically crashes.	True False
3	If a user types too few arguments but a program doesn't check for that, the program typically crashes.	True False

All command-line arguments are Strings. The (rather cumbersome) statement `age = Integer.parseInt(ageStr);` converts the ageStr String into an integer, assigning the result into int variable age. So string "12" becomes integer 12. parseInt() is a static method of the Integer class that returns the integer value of the input string.

Putting quotes around an argument allows an argument's string to have any number of spaces.

Figure 18.4.4: Quotes surround the single argument 'Mary Jo'.

```
java MyProg "Mary Jo" 50
```

P Participation Activity

18.4.4: String and integer command-line arguments.

#	Question	Your answer
1	What is the string in args[0] for the following: `java MyProg Amy Smith 19`	
2	What is the string in args[0] for the following: `java MyProg "Amy Smith" 19`	
3	Given the following code snippet, complete the assignment of userNum with args[0]. `public static void main(String[] args) {` `int userNum = 0;`	userNum = ;

Exploring further:

- Command-line arguments from Oracle's Java tutorials

Section 18.5 - Command-line arguments and files

The location of an input file or output file may not be known before writing a program. Instead, a program can use command-line arguments to allow the user to specify the location of an input file as shown in the following program. Assume two text files exist named "myfile1.txt" and "myfile2.txt" with the contents shown. The sample output shows the results when executing the program for each input file and for an input file that does not exist.

Figure 18.5.1: Using command-line arguments to specify the name of an input

file.

```
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.IOException;

public class FileReadNums {
   public static void main(String[] args) throws IOException {
      FileInputStream fileByteStream = null; // File input stream
      Scanner inFS = null;                   // Scanner object
      int fileNum1 = 0;                      // Data value from file
      int fileNum2 = 0;                      // Data value from file

      // Check number of arguments
      if (args.length != 1) {
         System.out.println("Usage: java FileReadNums inputFileName");
         return;
      }

      // Try to open the file
      System.out.println("Opening file " + args[0] + ".");

      fileByteStream = new FileInputStream(args[0]);
      inFS = new Scanner(fileByteStream);

      // File is open and valid if we got this far
      // myfile.txt should contain two integers, else problems
      System.out.println("Reading two integers.");
      fileNum1 = inFS.nextInt();
      fileNum2 = inFS.nextInt();

      // Done with file, so try to close it
      System.out.println("Closing file " + args[0] + "\n");
      fileByteStream.close(); // close() may throw IOException if fails

      // Output values read from file
      System.out.println("num1: " + fileNum1);
      System.out.println("num2: " + fileNum2);
      System.out.println("num1 + num2: " + (fileNum1 + fileNum2));

      return;
   }
}
```

```
> java FileReadNums myfile1.txt
Opening file myfile1.txt.
Reading two integers.
Closing file myfile1.txt.

num1: 5
num2: 10
num1 + num2: 15

...

> java FileReadNums myfile2.txt
Opening file myfile2.txt.
Reading two integers.
```

```
Closing file myfile2.txt.

num1: -34
num2: 7
num1 + num2: -27

...

> java FileReadNums myfile3.txt
Opening file myfile3.txt.
Exception in thread "main" java.io.FileNotFoundException: myfile3.txt (No such fil
    at java.io.FileInputStream.open(Native Method)
    at java.io.FileInputStream.(FileInputStream.java:137)
    at java.io.FileInputStream.(FileInputStream.java:96)
   at FileReadNums.main(FileReadNums.java:18)
```

P Participation Activity

18.5.1: Filename command-line arguments.

#	Question	Your answer
1	Assume a program has a single class called "MyProg", which contains main(). It takes in two command-line arguments, one for an input file and a second for an output file. Type a command to run the program with input file "infile.txt" and output file "out".	
2	For a program run as `java ProgName data.txt`, what is args[0]? Don't use quotes in your answer.	

Section 18.6 - Additional practice: Output art

The following is a sample programming lab activity; not all classes using a zyBook require students to fully complete this activity. No auto-checking is performed. Users planning to fully complete this program may consider first developing their code in a separate programming environment.

The following program prints a simple triangle.

P Participation Activity

18.6.1: Create ASCII art.

```java

public class PrintTriangle {
   public static void main (String [] args) {
      System.out.println("  *  ");
      System.out.println(" *** ");
      System.out.println("*****");

      return;
   }
}

```

Pre-enter any input for progra

Run

Create different versions of the program:

1. Print a tree by adding a base under a 4-level triangle:

```
   *
  ***
 *****
*******
  ***
```

2. Print the following "cat":

```
^   ^
 o o
=   =
 ---
```

3. Allow a user to enter a number, and then print the original triangle using that number instead of asterisks, as in:

```
  9
 999
99999
```

Pictures made from keyboard characters are known as ***ASCII art***. ASCII art can be quite intricate, and fun to make and view. Wikipedia: ASCII art provides examples. Doing a web search for "ASCII art (someitem)" can find ASCII art versions of an item. For example, searching for "ASCII art cat" turns up thousands of examples of cats, most much more clever than the cat above.

Section 18.7 - Additional practice: Grade calculation

The following is a sample programming lab activity; not all classes using a zyBook require students to fully complete this activity. No auto-checking is performed. Users planning to fully complete this program may consider first developing their code in a separate programming environment.

P Participation Activity | 18.7.1: Grade calculator.

The following incomplete program should compute a student's total course percentage based on scores on three items of different weights (%s):

- 20% Homeworks (out of 80 points)
- 30% Midterm exam (out of 40 points)
- 50% Final exam (out of 70 points)

Suggested (incremental) steps to finish the program:

1. First run it.
2. Next, complete the midterm exam calculation and run the program again. Use the constant variables where appropriate.
3. Then, complete the final exam calculation and run the program. Use the constant variables where appropriate.
4. Modify the program to include a quiz score out of 20 points. New weights: 10% homework, 15% quizzes, 30% midterm, 45% final. Run the program again.
5. To avoid having one large expression, introduce variables homeworkPart, quizPart,

midtermPart, and finalPart. Compute each part first; each will be a number between 0 and 1. Then combine the parts using the weights into the course value. Run the program again.

Reset

```

import java.util.Scanner;

public class GradeCalculator {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final double HOMEWORK_MAX = 80.0;
      final double MIDTERM_MAX  = 40.0;
      final double FINAL_MAX    = 70.0;
      final double HOMEWORK_WEIGHT = 0.20; // 20%
      final double MIDTERM_WEIGHT  = 0.30;
      final double FINAL_WEIGHT    = 0.50;

      double homeworkScore    = 0.0;
      double midtermScore     = 0.0;
      double finalScore       = 0.0;
      double coursePercentage = 0.0;

      System.out.println("Enter homework score:");
```

```
78 36 62
```

Run

Section 18.8 - Additional practice: Health data

The following is a sample programming lab activity; not all classes using a zyBook require students to fully complete this activity. No auto-checking is performed. Users planning to fully complete this program may consider first developing their code in a separate programming environment.

The following calculates a user's age in days based on the user's age in years.

P Participation Activity

18.8.1: Calculating user health data.

Reset

```

import java.util.Scanner;

public class HealthData {
   public static void main (String[] args) {
      Scanner scnr = new Scanner(System.in);
      int userAgeYears = 0;
      int userAgeDays  = 0;

      System.out.println("Enter your age in years: ");
      userAgeYears = scnr.nextInt();

      userAgeDays = userAgeYears * 365;

      System.out.println("You are " + userAgeDays + " days old.");

      return;
   }
}
```

19

Run

Create different versions of the program that:

1. Calculates the user's age in minutes and seconds.

2. Estimates the approximate number of times the user's heart has beat in his/her lifetime using an average heart rate of 72 beats per minutes.
3. Estimates the number of times the person has sneezed in his/her lifetime (research on the Internet to obtain a daily estimate).
4. Estimates the number of calories that the person has expended in his/her lifetime (research on the Internet to obtain a daily estimate). Also calculate the number of sandwiches (or other common food item) that equals that number of calories.
5. Be creative: Pick other health-related statistic. Try searching the Internet to determine how to calculate that data, and create a program to perform that calculation. The program can ask the user to enter any information needed to perform the calculation.

Section 18.9 - Additional practice: Tweet decoder

The following is a sample programming lab activity; not all classes using a zyBook require students to fully complete this activity. No auto-checking is performed. Users planning to fully complete this program may consider first developing their code in a separate programming environment.

The following program decodes a few common abbreviations in online communication as communications in Twitter ("tweets") or email, and provides the corresponding English phrase.

P Participation Activity 18.9.1:

```java

import java.util.Scanner;

public class TweetDecoder {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      String origTweet = "";

      System.out.println("Enter abbreviation from tweet:
      origTweet = scnr.next();

      if (origTweet.equals("LOL")) {
         System.out.println("LOL = laughing out loud");
      }
      else if (origTweet.equals("BFN")) {
         System.out.println("BFN = bye for now");
      }
      else if (origTweet.equals("FTW")) {
         System.out.println("FTW = for the win");
```

LOL

Run

Create different versions of the program that:

1. Expands the number of abbreviations that can be decoded. Add support for abbreviations you commonly use or search the Internet to find a list of common abbreviations.
2. For abbreviations that do not match the supported abbreviations, check for common misspellings. Provide a suggestion for correct abbreviation along with the decoded meaning. For example, if the user enters "LLO", your program can output "Did you mean LOL? LOL = laughing out loud".
3. Allows the user to enter a complete tweet (140 characters or less) as a single line of text. Search the resulting string for those common abbreviations and print a list of each abbreviation along with its decoded meaning.
4. Convert the user's tweet to a decoded tweet, replacing the abbreviations directly within the tweet.

Section 18.10 - Additional practice: Dice statistics

The following is a sample programming lab activity; not all classes using a zyBook require students to fully complete this activity. No auto-checking is performed. Users planning to fully complete this program may consider first developing their code in a separate programming environment.

Analyzing dice rolls is a common example in understanding probability and statistics. The following calculates the number of times the sum of two dice (randomly rolled) equals six or seven.

P Participation Activity

18.10.1: Dice rolls: Counting number of rolls that equals six or seven.

```java

import java.util.Scanner;
import java.util.Random;

public class DiceStats {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      Random randGen = new Random();
      int i = 0;          // Loop counter iterates numRo
      int numRolls = 0;   // User defined number of roll
      int numSixes = 0;   // Tracks number of 6s found
      int numSevens = 0;  // Tracks number of 7s found
      int die1 = 0;       // Dice values
      int die2 = 0;       // Dice values
      int rollTotal = 0;  // Sum of dice values


      System.out.println("Enter number of rolls: ");
      numRolls = scnr.nextInt();
```

10

Run

Create different versions of the program that:

1. Calculates the number of times the sum of the randomly rolled dice equals each possible value from 2 to 12.
2. Repeatedly asks the user for the number of times to roll the dice, quitting only when the user-entered number is less than 1. Hint: Use a while loop that will execute as long as numRolls is greater than 1. Be sure to initialize numRolls correctly.
3. Prints a histogram in which the total number of times the dice rolls equals each possible value is displayed by printing a character like * that number of times, as shown below.

Figure 18.10.1: Histogram showing total number of dice rolls for each possible value.

```
Dice roll histogram:

2:  ******
3:  ****
4:  ***
5:  ********
6:  *******************
7:  *************
8:  *************
9:  **************
10: ***********
11: *****
12: ****
```